CONTENTS

1 Modelling and Model Fitting

1.1 Matrix and Vector Operations

1.1.1 Vector

Definition 1. A column of real numbers is called a **vector**.

Example 1.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix} \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```
Python-code:
import numpy as np
a = np.array([[1],[-3],[2]])
print("a=",a)
One5 = np.ones([5])
print("One5=",One5)
Output:
a= [[ 1]
 [-3]
 [ 2]]
One5= [1. 1. 1. 1. 1.]
```

Since $\mathbf{y}$ has n elements it is said to have **order** (or dimension) n.

1.1.2 Matrix

Definition 2.
A rectangular array of elements with m rows and k columns is called an $m \times k$ **matrix.**

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}$$

This matrix is said to be of **order** (or dimension) $m \times k$, where

- m is the **row** order (dimension)
- k is the **column** order (dimension)
- a_{ij} is the (i, j) element of $\mathbf{A}$.

Example 2.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -2 \\ 0 & 4 & 5 \end{bmatrix} \quad \mathbf{I_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{J_3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
Python-code:
import numpy as np
A = np.array([[1, 3, -2], [0, 4, 5]])
I = np.identity(3)
One = np.ones(3)
J = np.ones((3,3))
print("A=", A)
print("I3=",I)
print("13=", One)
print("J3=",J)
output:
A= [[ 1  3 -2]
 [ 0  4  5]]
I3= [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
13= [1. 1. 1.]
J3= [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

1.1.3 Matrix Addition and Subtraction

Definition 3. Matrix addition
If $\mathbf{A}$ and $\mathbf{B}$ are both $m \times k$ matrices, then

$$\begin{aligned}\mathbf{C} &= \mathbf{A} + \mathbf{B} \\ &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mk} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1k}+b_{1k} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2k}+b_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mk}+b_{mk} \end{bmatrix}\end{aligned}$$

Notation:

$$C_{m\times k} = \{c_{ij}\} \text{ where } c_{ij} = a_{ij} + b_{ij}$$

Definition 4. Matrix subtraction

If $\mathbf{A}$ and $\mathbf{B}$ are $m \times k$ matrices, then $\mathbf{C} = \mathbf{A} - \mathbf{B}$ is defined by

$$\mathbf{C} = \{c_{ij}\} \text{ where } c_{ij} = a_{ij} - b_{ij} \ .$$

Example 3.

$$\begin{bmatrix} 3 & 6 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 7 & -4 \\ -3 & 2 \end{bmatrix} = \begin{bmatrix} 10 & 2 \\ -1 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix}$$

```
Python-code:
import numpy as np
A1 = np.array([[3, 6], [2,1]])
A2= np.array([[7, -4], [-3, 2]])
A3 = A1 + A2
print("A1+A2=",A3)

output:
A1+A2= [[10  2]
 [-1  3]]
import numpy as np
B1 = np.array([[1, -1], [1,1],[1,0]])
B2= np.array([[1, -1], [2, 0],[1,1]])
B3 = B1 - B2
print("B1-B2=",B3)

Output:
B1-B2= [[ 0  0]
 [-1  1]
 [ 0 -1]]
```

1.1.4 Scalar Multiplication

Definition 5. Scalar multiplication

Let a be a scalar and $\mathbf{B} = \{b_{ij}\}$ be an $m \times k$ matrix, then

$$a\,\mathbf{B} = \mathbf{B}\,a = \{a\,b_{ij}\}$$

Example 4.

$$2\begin{bmatrix} 2 & -1 & 3 \\ 0 & 4 & -2 \end{bmatrix} = \begin{bmatrix} 4 & -2 & 6 \\ 0 & 8 & -4 \end{bmatrix}$$

```
 Python-Code:
import numpy as np
B1 = np.array([[2, -1, 3], [0,4,-2]])
B2 = 2*B1
print("2*B1=",B2)
Output:
B2=2*B1= [[ 4 -2  6]
          [ 0  8 -4]]
```

1.1.5 Transpose

Definition 6. Transpose
The transpose of the $m \times k$ matrix $\mathbf{A} = \{a_{ij}\}$ is the $k \times m$ matrix with elements $\{a_{ji}\}$. The transpose of $\mathbf{A}$ is denoted by $\mathbf{A^T}$ (or $\mathbf{A'}$).

Example 5.

$$\mathbf{A} = \begin{bmatrix} 1 & 4 \\ 3 & 0 \\ -2 & 6 \end{bmatrix} \qquad \mathbf{A^T} = \begin{bmatrix} 1 & 3 & -2 \\ 4 & 0 & 6 \end{bmatrix}$$

```
R-code:
import numpy as np
B1 = np.array([[1,4], [3,0],[-2,6]])
B2 = B1.transpose()
print("B1^T = ",B2)

Output:
B1^T =
[[ 1  3 -2]
 [ 4  0  6]]
```

1.1.6 Inner Product

Definition 7. Inner product (crossproduct) of two vectors of order n

$$\mathbf{a}^T\mathbf{y} = [a_1, a_2, \cdots a_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = a_1y_1 + a_2y_2 + \cdots + a_ny_n$$

Example 6.

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 11 \\ 15 \\ 23 \end{bmatrix}$$

$$\mathbf{a^T y} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 11 \\ 15 \\ 23 \end{bmatrix} = 1(11) + 2(15) + 3(23) = 110$$

```
Python-codes:
import numpy as np
a = np.array([[1, 2, 3]])
y = np.array([[11, 15, 23]])
aty = np.inner(a,y)
print("a^Ty=",aty)
Output:
a^Ty= [[110]]
```

1.1.7 Matrix Multiplication

Definition 8. Matrix multiplication
The product of an $n \times k$ matrix $\mathbf{A}$ and a $k \times m$ matrix $\mathbf{B}$ is the $n \times m$ matrix $\mathbf{C} = \{c_{ij}\}$ with elements

$$c_{ij} = a_{i1}\, b_{1j} + a_{i2}\, b_{2j} + \cdots + a_{ik}\, b_{kj}$$

Example 7.

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & -2 \\ 1 & -1 & 4 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \quad \mathbf{C} = \mathbf{AB} = \begin{bmatrix} 1 & -3 \\ 4 & 11 \end{bmatrix}$$

```
Python-codes:
import numpy as np
A = np.array([[3, 0, -2], [1, -1, 4]])
B = np.array([[1,1],[1, 2],[1,3]])
C = A.dot(B)
print("C=AB",C)
Output:
C=AB=
[[ 1 -3]
 [ 4 11]]
```

Example 8.

Consider the following matrix:

$$\mathbf{A} = \begin{bmatrix} 17.7 & 16.7 & 15.7 \\ 12.1 & 11.1 & 13.1 \\ 12.1 & 14.4 & 10.1 \end{bmatrix}.$$

Use Python to find the $(3,3)$ element of $\mathbf{A^T A}$.

```
import numpy as np
A = np.array([[17.7,16.7,15.7],[12.1,11.1,13.1],
              [12.1,14.4,10.1]])
ATA = A.transpose().dot(A)
print("A^TA=",ATA)
ije = ATA[2,2]
print("(2,2) element of A = ", ije)
Output:
A^TA= [[606.11 604.14 558.61]
 [604.14 609.46 553.04]
 [558.61 553.04 520.11]]
(3,3) element of A =  520.1099999999999
```

1.1.8 Elementwise multiplication

Definition 9. Elementwise multiplication of two matrices

$$\mathbf{A} \# \mathbf{B} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{km} \end{bmatrix} \# \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & & \vdots \\ b_{k1} & \cdots & b_{km} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & \cdots & a_{1m}\,b_{1m} \\ \vdots & & \vdots \\ a_{k1}\,b_{k1} & \cdots & a_{km}\,b_{km} \end{bmatrix}$$

Example 9.

$$\begin{bmatrix} 3 & 1 \\ 2 & 4 \\ 0 & 6 \end{bmatrix} \# \begin{bmatrix} 1 & -5 \\ -3 & 4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & -5 \\ -6 & 16 \\ 0 & 12 \end{bmatrix}$$

```
Python-codes:
import numpy as np
A = np.array([[3, 1], [2, 4],[0,6]])
B = np.array([[1,-5],[-3, 4],[-2,2]])
C = A*B
print("C=AB=",C)
Output:
C=AB=
[[ 3 -5]
 [-6 16]
 [ 0 12]]
```

1.1.9 Kronecker Product

Definition 10. Kronecker product of two matrices

$$\mathbf{A}_{k\times m} \otimes \mathbf{B}_{n\times s} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1m}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2m}\mathbf{B} \\ \vdots & \vdots & & \vdots \\ a_{k1}\mathbf{B} & a_{k2}\mathbf{B} & \cdots & a_{km}\mathbf{B} \end{bmatrix}$$

Example 10.

$$\begin{bmatrix} 2 & 4 \\ 0 & -2 \\ 3 & -1 \end{bmatrix} \otimes \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 6 & 20 & 12 \\ 4 & 2 & 8 & 4 \\ 0 & 0 & -10 & -6 \\ 0 & 0 & -4 & -2 \\ 15 & 9 & -5 & -3 \\ 6 & 3 & -2 & -1 \end{bmatrix}$$

$$\mathbf{a} \otimes \mathbf{y} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_1\, y_1 \\ a_1\, y_2 \\ a_2\, y_1 \\ a_2\, y_2 \\ a_3\, y_1 \\ a_3\, y_2 \end{bmatrix}$$

```
import numpy as np
A = np.array([[2, 4], [0, -2],[3,-1]])
B = np.array([[5,3],[2, 1]])
C = np.kron(A,B)
print("C=AB =", C)
print(C)
Output:
C=AB =
[[ 10   6  20  12]
 [  4   2   8   4]
 [  0   0 -10  -6]
 [  0   0  -4  -2]
 [ 15   9  -5  -3]
```

1.1.10 Determinant

Definition 11. The **determinant** of an $n \times n$ matrix $\mathbf{A}$ is

$$|\mathbf{A}| = \sum_{j=1}^{n} a_{ij}(-1)^{i+j}|M_{ij}| \quad \text{for any row } i$$

or

$$|\mathbf{A}| = \sum_{i=1}^{n} a_{ij}(-1)^{i+j}|M_{ij}| \quad \text{for any column } j$$

where M_{ij} is the "minor" for a_{ij} obtained by deleting the i^{th} row and j^{th} column from A.

Example 11.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$|\mathbf{A}| = a_{11}(-1)^{1+1}|a_{22}| + a_{12}(-1)^{1+2}|a_{21}|$$

then $\begin{vmatrix} 7 & 2 \\ 4 & 5 \end{vmatrix} =$

Example 12.

$$|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$= a_{11}(-1)^{1+1} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{12}(-1)^{1+2} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix}$$

$$+ a_{13}(-1)^{1+3} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$\text{then } \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} =$$

```
Python Code:
import numpy as np
A = np.array([[7, 2], [4, 5]])
B = np.array([[1,2,3],[4, 5,6],[7,8,9]])
DA = np.linalg.det(A)
DB = np.linalg.det(B)
print("DA=|A|=", DA)
print("DB=|B|=",DB)
Output:
DA=|A|=  27.0
DB=|B|=  6.66133814775094e-16
```

1.1.11 Inverse

Definition 12. The **identity matrix**, denoted by $\mathbf{I}$, is a $k \times k$ matrix of the form

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Definition 13. The **inverse** of a square, non-singular matrix $\mathbf{A}$ is the matrix, denoted by $\mathbf{A}^{-1}$, such that

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = I$$

Example 13.

Determine the inverse of

$$\mathbf{A} = \begin{pmatrix} 7 & 2 \\ 4 & 5 \end{pmatrix} = \text{and } \mathbf{B} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
R-codes:
import numpy as np
A = np.array([[7, 2], [4, 5]])
B = np.array([[1,2,3],[4, 5,6],[7,8,9]])
IA = np.linalg.inv(A)
IB = np.linalg.inv(B)
print("IA=A^(-1)=", IA)
print("IB=B^(-1)=",IB)
Output:
IA=A^(-1)= [[ 0.18518519 -0.07407407]
 [-0.14814815  0.25925926]]
IB=B^(-1)= [[-4.50359963e+15  9.00719925e+15 -4.50359963e+15]
 [ 9.00719925e+15 -1.80143985e+16  9.00719925e+15]
 [-4.50359963e+15  9.00719925e+15 -4.50359963e+15]]
```

1.1.12 Trace

Definition 14. The **trace** of a $k \times k$ matrix $\mathbf{A} = \{a_{ij}\}$ is the sum of the diagonal elements:

$$tr(\mathbf{A}) = \sum_{j=1}^{k} a_{jj}$$

Example 14.

Consider the following matrix:

$$\mathbf{A} = \begin{pmatrix} 179 & 169 & 159 \\ 124 & 114 & 134 \\ 125 & 170 & 105 \end{pmatrix}.$$

Use Python to find the trace of matrix $\mathbf{A}$.

```
import numpy as np
A = np.array([[179,169,159],[124,114,134],[125,170,105]])
TRA = np.trace(A)
print("Trace of A =", TrA)
Output:
Trace of A = 398
```

1.1.13 Eigenvalues and Eigenvectors

Definition 15. For a $k \times k$ matrix $\mathbf{A}$, the scalars $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ satisfying the polynomial equation

$$|\mathbf{A} - \lambda\, \mathbf{I}| = 0$$

are called the eigenvalues (or characteristic roots) of $\mathbf{A}$.

Definition 16. Corresponding to any eigenvalue λ_i is an eigenvector (or characteristic vector) $\mathbf{u}_i \neq \mathbf{0}$ satisfying

$$\mathbf{A}\, \mathbf{u}_i = \lambda_i\, \mathbf{u}_i.$$

Example 15. Find the eigenvalue and eigenvector of

$$\mathbf{A} = \begin{bmatrix} 1.96 & 0.72 \\ 0.72 & 1.54 \end{bmatrix}$$

```
 Python codes:
import numpy as np
A = np.array([[1.96, .72], [.72,1.54]])
w, v = np.linalg.eig(A)
print("w=", w)
print("v=",v)
Output:
w=
[2.5 1. ]
v=
[[ 0.8 -0.6]
 [ 0.6  0.8]]
```

1.2 System of Linear Equations

A system of linear equations (or linear system) is a collection of one or more linear equations involving the same set

of variables. For example,

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

is a system of three equations in the three variables x, y, z. A solution to a linear system is an assignment of values to the variables such that all the equations are simultaneously satisfied. A solution to the system above is given by $x = 1, y = -2, z = -2$ since it makes all three equations valid. The word "system" indicates that the equations are to be considered collectively, rather than individually.

A general system of m linear equations with n unknowns can be written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m_1}x_1 + a_{m_2}x_2 + \cdots + a_{m_n}x_n &= b_m \end{aligned}$$

where $x_1, x_2, \ldots, x_n$ are the unknowns, $a_{11}, a_{12}, \ldots, a_{mn}$ are the coefficients of the system, and $b_1, b_2, \ldots, b_n$ are the constant terms.

1.2.1 Solving System of Linear Equations Using Matrix Inversion

The system of linear equations can be expressed in matrix equation of the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x}$ is a column vector with n entries, and $\mathbf{b}$ is a column vector with m entries.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

If the matrix $\mathbf{A}$ is square (has m rows and $n = m$ columns) and has full rank (all m rows are independent), then the system has a unique solution given by

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Example 16.

Solve the system of linear equations below using matrix inversion.

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

```
Python code:
import numpy as np
A = np.array([[3, 2,-1], [2,-2,4],[-1,1./2,-1]])
b = np.array([[1],[-2],[0]])
IA = np.linalg.inv(A)
print("A^(-1}=",IA)
x = IA.dot(b)
print("x=A^(-1)b",x)
Output:
A^(-1}= [[ 2.22044605e-16 -5.00000000e-01 -2.00000000e+00]
 [ 6.66666667e-01  1.33333333e+00  4.66666667e+00]
 [ 3.33333333e-01  1.16666667e+00  3.33333333e+00]]
x=A^(-1)b [[ 1.]
 [-2.]
 [-2.]]
```

1.2.2 Solving System of Linear Equations Using linalg's Solve Routine

The system of linear equations can be also solve using linalg's Routine in numpy package. Instead of solving $\mathbf{x} = \mathbf{A^{-1}b}$. We can obtained the solution by using the linalg's solve routine.

Example 17.
Solve the system of linear equations below using linalg's solve routine.

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

```
Python code:
import numpy as np
A = np.array([[3, 2,-1], [2,-2,4],[-1,1./2,-1]])
b = np.array([[1],[-2],[0]])
x = np.linalg.solve(A,b)
print("x=",x)
Output:
x [[ 1.]
 [-2.]
 [-2.]]
```

1.3 Regression Analysis

1.3.1 What Is Regression?

Regression searches for relationships among variables. For example, you can observe the selling price of properties and try to understand how the area of living space on the properties, the appraised home value on the properties, and appraised land value of the properties, the selling price depend on the features, such as the area of living space on the property, the appraised home value on the property, and appraised land value of the property, and so on.

This is a regression problem where data related to each property represent one observation. The presumption is that the area of living space on the property, the appraised home value on the property, and appraised land value of the property are the independent features, while the selling price depends on them.

Generally, in regression analysis, you usually con-

sider some phenomenon of interest and have a number of observations. Each observation has two or more features. Following the assumption that (at least) one of the features depends on the others, you try to establish a relation among them. In other words, you need to find a function that maps some features or variables to others sufficiently well. The dependent features are called the **dependent variables, outputs, or responses**. The independent features are called the **independent variables, inputs, or predictors**.

Regression problems usually have one continuous and unbounded dependent variable. The inputs, however, can be continuous, discrete, or even categorical data such as gender, nationality, brand, and so on.

It is a common practice to denote the outputs with y and inputs with x. If there are two or more independent variables, they can be represented as the vector $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the

number of inputs.

1.3.2 When Do You Need Regression?

Typically, you need regression to answer whether and how some phenomenon influences the other or how several variables are related. For example, you can use it to determine if and to what extent the area of living space on the property, the appraised home value on the property, and appraised land value of the property impact selling price.

Regression is also useful when you want to forecast a response using a new set of predictors. For example, you could try to predict electricity consumption of a household for the next hour given the outdoor temperature, time of day, and number of residents in that household.

Regression is used in many different fields: economy, computer science, social sciences, and so on. Its importance rises every day with the availability of large amounts of data and increased aware-

ness of the practical value of data.

1.3.3 Problem Formulation

When implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and x:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \epsilon.$$

This equation is the regression equation. $\beta_0, \beta_1, \ldots, \beta_r$ are the regression coefficients, and ϵ is the random error.

Linear regression calculates the estimators of the regression coefficients, denoted with $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_r$. They define the estimated regression function $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_r x_r$. This function should capture the dependencies between the inputs and output sufficiently well.

The estimated or predicted response, $\hat{y}_i$, for each observation $i = 1, \ldots, n$, should be as close as

possible to the corresponding actual response y_i. The differences $y_i - \hat{y}_i$ for all observations $i = 1, \ldots, n$, are called the residuals or errors. Regression is about determining the best predicted weights, that is the weights corresponding to the smallest residuals.

To get the best weights, you usually minimize the sum of squared residuals(errors) (SSE) for all observations $i = 1, \ldots, n$, $\text{SSE} = \sum_i (y_i - \hat{y}_i)^2$. This approach is called the method of ordinary least squares(OLS).

1.3.4 Regression Performance

The variation of actual responses $y_i, i = 1, \ldots, n$ occurs partly due to the dependence on the predictors x_i. However, there is also an additional inherent variance of the output.

The coefficient of determination, denoted as R^2, tells you which amount of variation in y can be explained by the dependence on x using the particular regression model. Larger R^2 indicates a

better fit and means that the model can better explain the variation of the output with different inputs.
The value $R^2 = 1$ corresponds to $SSE = 0$, that is to the perfect fit since the values of predicted and actual responses fit completely to each other. Thus, the larger the R^2, the better fit the data to the model. However, R^2 increases as the number parameters increases. Thus we usually will use adjusted R^2 to choose the appropriate model. Another two quantity that use to make decision on model selection are Aikaike Information Criterion(AIC) and Bayesisn Information Criterion(BIC).

1.3.5 Simple Linear Regression

Simple or single-variate linear regression is the simplest case of linear regression with a single independent variable, $\mathbf{x} = x$.
When implementing simple linear regression, you typically start with a given set of input-output (x, y) pairs. These pairs are your observations.

The estimated regression function has the equation $y = \beta_0 + \beta_1 x$. Your goal is to calculate the optimal values of the predicted weights β_0 and β_1 that minimize Sum of Square of Error (SSE) and determine the estimated regression function. The value of β_0, also called the intercept, shows the point where the estimated regression line crosses the y axis. It is the value of the estimated response $y = 0$ for $x = 0$. The value of β_1 determines the slope of the estimated regression line.

1.3.6 Multiple Linear Regression

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

If there are just two independent variables, the estimated regression function is $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$. It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\beta}_2$ such that this plane is as close as possible to the

actual responses and yield the minimal SSE.
The case of more than two independent variables is similar, but more general. The estimated regression function is $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots \hat{\beta}_r x_r$ and there are $r + 1$ weights to be determined when the number of inputs is r.

1.3.7 Polynomial Regression

You can regard polynomial regression as a generalized case of linear regression. You assume the polynomial dependence between the output and inputs and, consequently, the polynomial estimated regression function.
In other words, in addition to linear terms like $\hat{\beta}_1 x_1$, your regression function y can include nonlinear terms such as $\hat{\beta}_2 x_1^2$, $\hat{\beta}_1 x_1^3$, or even $\hat{\beta}_4 x_1 x_2$, $\hat{\beta}_5 x_1 x_3$ and so on.
The simplest example of polynomial regression has a single independent variable, and the estimated regression function is a polynomial of degree 2: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \beta_2 x^2$.

Keeping this in mind, compare the previous regression function with the function $\hat{y} = \hat{\beta}_0 + \beta_1 x + \beta_2 x^2$ used for linear regression. They look very similar and are both linear functions of the unknowns $\hat{\beta}_0$, $\hat{\beta}_1$, and $\hat{\beta}_2$. This is why you can solve the polynomial regression problem as a linear problem with the term x^2 regarded as an input variable.

In the case of two variables and the polynomial of degree 2, the regression function has this form: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_1^2 + \beta_4 x_1 x_2 + \hat{\beta}_5 x_2^2$. The procedure for solving the problem is identical to the previous case. You apply linear regression for five inputs: $x_1, x_2, x_1^2, x_1 x_2, x_2^2$, and X_2^2. What you get as the result of regression are the values of six weights which minimize SSE: $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3, \hat{\beta}_4$, and $\hat{\beta}_5$.

1.4 Ordinary Least Squares Estimation

For the linear model with

$$\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

we have

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1r} \\ X_{21} & X_{22} & \cdots & X_{2r} \\ \vdots & \vdots & & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{nr} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_r \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

and

$$\begin{aligned} y_i &= \boldsymbol{\beta}_1\,\mathbf{x}_{i1} + \boldsymbol{\beta}_2\,\mathbf{x}_{i2} + \cdots + \boldsymbol{\beta}_r\,\mathbf{x}_{ir} + \boldsymbol{\epsilon}_i \\ &= \mathbf{X}_i^T\,\boldsymbol{\beta} + \boldsymbol{\epsilon}_i \end{aligned}$$

where $\mathbf{X}_i^T = (\mathbf{x}_{i1}, \mathbf{x}_{i2}, \cdots, \mathbf{x}_{ir})$ is the i-th row of the model matrix $\mathbf{X}$.

Definition 17.
For a linear model with $\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \boldsymbol{\epsilon}$ any vector $\mathbf{b}$ that minimizes the sum of squared residuals

$$\begin{aligned} Q(\mathbf{b}) &= \textstyle\sum_{i=1}^{n} (y_i - \mathbf{X}_i^T\mathbf{b})^2 \\ &= (\mathbf{y} - X\mathbf{b})^T(\mathbf{y} - \mathbf{X}\mathbf{b}) \end{aligned}$$

is an ordinary least squares (OLS) estimator for $\boldsymbol{\beta}$.

For $j = 1, 2, \ldots, r$, solve

$$0 = \frac{\partial Q(\mathbf{b})}{\partial b_j} = 2 \sum_{i=1}^{n} (y_i - \mathbf{X}_i^T \mathbf{b}) \, X_{ij}$$

Dividing by 2, we have

$$0 = \sum_{i=1}^{n} (y_i - \mathbf{X}_i^T \mathbf{b}) \, X_{ij} \quad j = 1, 2, \ldots, r$$

These equations are expressed in matrix form as

$$\begin{aligned} \mathbf{0} &= \mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{b}) \\ &= \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{b} \end{aligned}$$

or

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$$

These are often called the "normal" equations.

If $\mathbf{X}_{n \times r}$ has full column rank, i.e., $\text{rank}(\mathbf{X}) = k$, then

$$(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

and

$$\boxed{\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

is the unique solution to the normal equations.

If $\text{rank}(\mathbf{X}) < k$, then the solution does not exist.

Example 18. Regression Analysis:Yield of a chemical process

Yield (%)	Temperature (^oF)	Time (hr)
y	x_1	x_2
77	160	1
82	165	3
84	165	2
89	170	1
94	175	2

Multiple linear regression model

$$y_i = \beta_0 + \beta_1\, x_{i1} + \beta_2\, x_{i2} + \epsilon_i$$

$$i = 1, 2, 3, 4, 5$$

Matrix formulation:

Example 19. A local Bank wants to estimate the number of foreclosures per 1000 houses sold. Bank officials think that the number of foreclosures is related to the size of the down payment made by house buyers. The following table contains foreclosure and down payment data.

Down Payment Size (% of purchase price)	Number of Foreclosures per 1000 houses
10	36
20	9
14	28
12	25
18	12
16	16

You fit the above data to $Y = \beta_0 + \beta_1 X + \epsilon$, where Y is the number of foreclosures per 1000 houses sold, and X is the size of the down payment made by house buyers. Write the Python commands and output using matrix formulation to obtain the estimate of β_0 and β_1.

```
import numpy as np
y = np.array([36,9,28,25,12,16])
x0 = np.array(np.repeat(1,6)).reshape(-1,1)
x1 = np.array([10,20,14,12,18,16]).reshape(-1,1)
x = np.hstack((x0, x1))
xt = x.transpose()
xtx = xt.dot(x)
```

```
Ixtx = np.linalg.inv(xtx)
xty = xt.dot(y)
b = Ixtx.dot(xty)
print("b=",b)
Output:
b= [60.85714286 -2.65714286]
```

Example 20. Refer to example 18, obtain the OLS estimate, **b**.

Python code:

```
import numpy as np
y = [77,82,84,89,94]
x0 = np.array(np.repeat(1,5)).reshape(-1,1)
x1 = np.array([160,165,165,170,175]).reshape(-1,1)
x2 = np.array([1,3,2,1,2]).reshape(-1,1)
x = np.hstack((x0, x1,x2))
xt = x.transpose()
xty = xt.dot(y)
xtx = xt.dot(x)
Ixtx = np.linalg.inv(xtx)
b = Ixtx.dot(xty)
print("b=",b)
```

Output:

```
b= [-105.22222222    1.14444444   -0.38888889]
```

Example 21.

A researcher believes that the number of days the ozone levels exceeded 0.2ppm (y) depends on the seasonal meteorological index (x). The following table gives the data.

Index	16.6	16.5	17.6	18.0	16.4	16.3	14.4	16.7	17.5	16.9
Days	90	109	113	118	83	83	58	79	72	57

You fit the above data to $Y = \beta_0 + \beta_1 X + \epsilon$, where Y is the number of days the ozone levels exceeded 0.2ppm, and X is the seasonal meteorological index.

(a) Write the model above in the form $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$

(b) Write the Python commands and output using matrix formulation to obtain the estimate of β_0 and β_1.

(c) You are given that $R^2 = \frac{SSR}{SST}$ and adjusted R^2, $R^2_{Adj} = 1 - \frac{SSE/(n-p)}{SST/(n-1)}$, where

- n is the number of observations.
- p is the number of parameters in the model.
- $SSR = \widehat{\boldsymbol{\beta}}^T \mathbf{X}^\mathbf{T}\mathbf{y} - \frac{1}{n}\mathbf{y}^\mathbf{T}\mathbf{J}\mathbf{y}$, where $\mathbf{J}$ is an $n \times n$ matrix of one.
- $SSE = \mathbf{y}^\mathbf{T}\mathbf{y} - \widehat{\boldsymbol{\beta}}^T \mathbf{X}^\mathbf{T}\mathbf{y}$.
- $SST = \mathbf{y}^\mathbf{T}\mathbf{y} - \frac{1}{n}\mathbf{y}^\mathbf{T}\mathbf{J}\mathbf{y}$.

Write the Python commands and output to calculated R^2 and adjusted R^2.

(b)
```
import numpy as np
y = np.array([90,109,113,118,83,83,58,79,72,57])
x = np.array([[1,16.6],[1,16.5],[1,17.6],[1,18.0],[1,16.4],
              [1,16.3],[1,14.4],[1,16.7],[1,17.5],[1,16.9]])
xtx = x.transpose().dot(x)
xty = x.transpose().dot(y)
b = np.linalg.inv(xtx).dot(xty)
print(b)
Output:
[-123.96191127835846 12.592085756641836]
```

(b)
```
import numpy as np
y = np.array([90,109,113,118,83,83,58,79,72,57])
x = np.array([[1,16.6],[1,16.5],[1,17.6],[1,18.0],[1,16.4],
              [1,16.3],[1,14.4],[1,16.7],[1,17.5],[1,16.9]])
n = len(y)
p = 2
xtx = x.transpose().dot(x)
xty = x.transpose().dot(y)
b = np.linalg.inv(xtx).dot(xty)
print("beta hat =", b)
yty = y.transpose().dot(y)
bxty = b.transpose().dot(xty)
J = np.ones((n,n))
ytJy = y.transpose().dot(J).dot(y)
SSR = bxty-ytJy/n
SSE = yty-bxty
SST = yty-ytJy/n
print("SSE=", SSE)
print("SSR=", SSR)
print("SST=",SST)
RSq = SSR/SST
AdjRSq = 1-(SSE/(n-p))/(SST/(n-1))
print("Rsq=",RSq)
print("AdjRsq=", AdjRSq)
Output:
```

```
beta hat = [-123.96191127835846 12.592085756641836]
SSE= 2795.1818907569686
SSR= 1390.4181092430372
SST= 4185.600000000006
RSq= 0.3321908709009545
AdjRSq= 0.24871472976357378
```

1.5 Linear Regression With Python

1.5.1 Simple and Multiple Regression

Step 1: Import packages

First you need to do some imports. In addition to numpy, you need to import statsmodels.api:

```
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

Now you have the packages you need.

Step 2: Provide data and transform inputs

The second step is defining data to work with. The regressors,x and predictor, y should be arrays.

```
y = np.array([5, 20, 14, 32, 22, 38])
n = len(y)
x0 = np.array(np.repeat(1,n)).reshape((-1, 1))
x1 = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
x = np.hstack((x0, x1))
```

Now, you have two arrays: the x and y. You should call .reshape() on x because this array is required to be two-dimensional, or to be more precise, to have one column and as many rows as necessary.

That's exactly what the argument (-1, 1) of .reshape() specifies.

Step 3: Create a model and fit it
The regression model based on ordinary least squares is an instance of the class

statsmodels.regression.linear_model.OLS.

This is how you can obtain one:

```
model = sm.OLS(y, x)
```

Once your model is created, you can apply .fit() on it:

```
results = model.fit()
```

By calling .fit(), you obtain the variable results, which is an instance of the class statsmodels.regression.

linear_model.RegressionResultsWrapper.

This object holds a lot of information about the regression model.

Step 4: Get results

You can call .summary() to get the table with the results of linear regression:

```
print(results.summary())
```

```
 OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.716
Model:                            OLS   Adj. R-squared:                  0.645
Method:                 Least Squares   F-statistic:                     10.08
Date:                Fri, 29 Oct 2021   Prob (F-statistic):             0.0337
Time:                        17:19:23   Log-Likelihood:                -19.071
No. Observations:                   6   AIC:                             42.14
Df Residuals:                       4   BIC:                             41.73
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.6333      5.872      0.959      0.392     -10.670      21.936
x1             0.5400      0.170      3.175      0.034       0.068       1.012
==============================================================================
Omnibus:                        nan   Durbin-Watson:                   3.606
Prob(Omnibus):                  nan   Jarque-Bera (JB):                0.651
Skew:                         0.008   Prob(JB):                        0.722
Kurtosis:                     1.387   Cond. No.                         69.8
==============================================================================
```

Step 5: Predict response

You can obtain the predicted response on the input values used for creating the model using .fittedvalues or .predict() with the input array as the argument:

```
print('predicted response:',results.fittedvalues)
print('predicted response:',results.predict(x))
```

```
predicted response: [ 8.33333333 13.73333333 19.13333333 24.53333333
 29.93333333 35.33333333]
```

This is the predicted response for known inputs. If you want predictions with new regressors, you can also apply .predict() with new data as the argument:

```
x_new = np.array([1, 20])
y_new = results.predict(x_new)
print("y_new=",y_new)
```

Example 22.

Consider the data shown below:

y	x	y	x
31	14	26	11
22	8	23	9
23	9	16	4
20	7	31	14
28	12	22	8

You fit the above data to $Y = \beta_0 + \beta_1 X + \epsilon$.

1. State the estmated regression function.
2. Obtain a prediction for a new observation y_h when $x_h = 17.0$.

```
Python code:
import numpy as np
import statsmodels.api as sm
y = np.array([31,22,23,20,28,16,31,222623])
n = len(y)
x0 = np.array(np.repeat(1,n)).reshape((-1,1))
x1 = np.array([14,8,9,7,12,4,14,8119]).reshape((-1,1))
x = np.hstack((x0,x1))
model = sm.OLS(y,x)
results = model.fit()
beta = [9.6726 1.5133]
print("Beta hat = ",beta)
#print(results.summary())
xnew = np.array([1,17.0])
ynew = results.predict(17.0)
print("y hat = ",ynew)
Output:
```

```
Beta hat = [9.672566371681416 1.5132743362831858]
y_new= [[35.39823009]]
```

Example 23.

The data file VegiB.csv record a flavor and texture score, Y (between 0 and 100) for 12 brands of meatless hamburgers along with the price, X_1, number of calories, X_2, amount of fat, X_3, and amount of sodium per burger, X_4. Assuming that regression model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \epsilon$ is appropriate.

1. Fit a regression model to the data. State the estimated regression function.
2. Determine the predicted value for the mean score of flavor and texture for all brands of meatless hamburgers with the price = 88, number of calories = 144, amount of fat = 4, and amount of sodium per burger = 494.

1.5.2 Polynomial Regression

Implementing polynomial regression is very similar to linear regression. There is only one extra step: you need to transform the array of inputs to include non-linear terms such as x^2.

Step 1 Import packages and classes

In addition to numpy and statsmodels, you should also import the class

PolynomialFeatures from sklearn.preprocessing:

```
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

Step 2 Provide data and transform input data

The input and output and is the same as in the case of linear regression:

```
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([15, 11, 2, 8, 25, 32])
```

As you've seen earlier, you need to include x^2 (and perhaps other terms) as additional features when

implementing polynomial regression. For that reason, you should transform the input array x to contain the additional column(s) with the values of x^2 (and eventually more features). It's possible to transform the input array in several ways (like using insert() from numpy), but the class PolynomialFeatures is very convenient for this purpose. Let's create an instance of this class:

```
transformer = PolynomialFeatures(degree=2,
                                 include\_bias=False)
```

The variable transformer refers to an instance of PolynomialFeatures which you can use to transform the input x.

You can provide several optional parameters to PolynomialFeatures:

- degree is an integer (2 by default) that represents the degree of the polynomial regression function.
- interaction_only is a Boolean (False by default) that decides whether to include only interaction features (True) or all features (False).
- include_bias is a Boolean (True by default) that decides whether to include the bias (intercept) column of ones (True) or not (False).

Before applying transformer, you need to fit it with .fit():

```
transformer.fit(x)
```

Once transformer is fitted, it's ready to create a new, modified input. You apply .transform() to do that:

```
x_ = transformer.transform(x)
```

You can also use .fit_transform() to replace the three previous statements with only one:

```
x_ = PolynomialFeatures(degree=2,
                        include_bias=False).fit_transform(x)
```

That's fitting and transforming the input array in one statement with .fit_transform().

Example 24.

Consider the data shown below:

y	x	y	x
1,144,490	32	428,736	25
176,732	20	258,005	22
176,732	20	1,008,642	31
92,775	17	23,397	12
672,787	28	214,493	21
11,404	10	214,493	21
1,293,618	33	773,565	29
144,191	19	92,775	17

Fit the polynomial regression models up to order 4, then answer the following questions:

(a) Fill in the table below:

Polynomial model	Adjusted R^2	AIC	BIC
First order			
Second order			
Third order			
Fourth order			

(b) Determine a model that best fit the data based on the adjusted R^2.

(c) Write down the equation of the best fitted curve that you have selected from part (b)

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
y = np.array([1,144,490,176,732,176,732,92,775,672,787,11,404,1,293,618,1
              428,736,258,005,1,008,642,23,397,214,493,214,493,773,565,92
print(y)
x = np.array([32,20,20,17,28,10,33,19,
              25,22,31,12,21,21,29,17]]).reshape((-1,1))
x1st = PolynomialFeatures(degree=1, include_bias=True).fit_transform(x)
x2nd = PolynomialFeatures(degree=2, include_bias=True).fit_transform(x)
x3rd = PolynomialFeatures(degree=3, include_bias=True).fit_transform(x)
x4th = PolynomialFeatures(degree=4, include_bias=True).fit_transform(x)
model1 = sm.OLS(y, x1st)
results1 = model1.fit()
model2 = sm.OLS(y, x2nd)
results2 = model2.fit()
model3 = sm.OLS(y, x3rd)
results3 = model3.fit()
model4 = sm.OLS(y, x4th)
results4 = model4.fit()
print("Adj. R-squared model1 = ", results1.rsquared_adj)
print("AIC model1 = ", results1.aic)
print("BIC model1 = ", results1.bic)
print("Adj. R-squared model2 = ", results2.rsquared_adj)
print("AIC model2 = ", results2.aic)
print("BIC model2 = ", results2.bic)
print("Adj. R-squared model3 = ", results3.rsquared_adj)
print("AIC model3 = ", results3.aic)
print("BIC model3 = ", results3.bic)
print("Adj. R-squared model4 = ", results4.rsquared_adj)
print("AIC model4 = ", results4.aic)
print("BIC model4 = ", results4.bic)
print("Beta hat = ", results4.params)
#print("results1",results1.summary())
#print("Results2",results2.summary())
```

```
#print("Results3",results3.summary())
#print("results4",results4.summary())
Output:
Adj. R-squared model1 =  0.861
AIC model1 =  430.2
BIC model1 =  431.7
Adj. R-squared model2 =  0.996
AIC model2 =  373.2
BIC model2 =  375.5
Adj. R-squared model3 =  1.0
AIC model3 =  288.3
BIC model3 =  291.4
Adj. R-squared model4 =  1.0
AIC model4 =  10.607
BIC model4 =  14.47
Beta hat =  [0.7636 0.0454 0.2346 0.6794 1.0700]
```

1.6 Visualizing Data in Python

A scatter plot is a visual representation of how two variables relate to each other. You can use scatter plots to explore the relationship between two variables, for example by looking for any correlation between them.
One of the most popular modules is **Matplotlib** and its submodule **pyplot**, often referred to using the alias **plt**. Matplotlib provides a very versatile tool called **plt.scatter()** that allows you to create both basic and more complex scatter plots.
A cafe sells six different types of bottled orange drinks. The owner wants to understand the relationship between the price of the drinks and how many of each one he sells, so he keeps track of how many of each drink he sells every day. You can visualize this relationship as follows:

```
import matplotlib.pyplot as plt
price = [2.50, 1.23, 4.02, 3.25, 5.00, 4.40]
sales_per_day = [34, 62, 49, 22, 13, 19]
plt.scatter(price, sales_per_day)
plt.title("Sales_per_day vs Prices for Orange Drinks")
plt.xlabel("Price")
plt.ylabel("Sales_per_day")
plt.show()
```

Here's the output from this code: